Correction Working with Inheritance - 0

C - Compilation error for Dog(String) constructor

abstract class can have constructors and it also possible to have abstract class without any abstract method. So, there is no issue with Animal class.

Java compiler adds super(); as the first statement inside constructor, if call to another constructor using this(…) or super(…) is not available.

Inside Animal class Constructor, compiler adds super(); => Animal(String name) { super(); this.name = name; }, super() in this case invokes the no-arg constructor of Object class and hence no compilation error here.

Compiler changes Dog(String) constructor to: Dog(String breed) { super(); this.breed = breed; }. No-arg constructor is not available in Animal class and as another constructor is provided, java compiler doesn't add default constructor. Hence Dog(String) constructor gives compilation error.

There is no issue with Dog(String, String) constructor.

Correction Working with Inheritance - 1

D -

```
Replace /*INSERT-1*/ with:
super(capacity);
this.type = type;
Replace /*INSERT-2*/ with:
super(0);
```

Java compiler adds super(); as the first statement inside constructor, if call to another constructor using this(…) or super(…) is not available.

Compiler adds super(); as the first line in OTG's constructor: OTG(int capacity, String type) { super(); } but PenDrive class doesn't have a no-arg constructor and that is why OTG's constructor causes compilation error.

For the same reason, OTG(String make) constructor also causes compilation error.

To correct these compilation errors, parent class constructor should be invoked by using super(int); This would resolve compilation error.

Remember: Constructor call using this(…) or super(…) must be the first statement inside the constructor.

In the main(String[]) method, OTG(int, String) constructor is invoked, which means, we OTG(String) constructor will not be executed. So, to solve the complilation error in OTG(String) constructor, super(0); or super(128); both will work and these will not affect the expected output.

We have to make changes in OTG(int, String) constructor such that on execution, output is 128:TYPE-C.

super(capacity); will only assign value to capacity property, to assign value to type another statement is needed.

this.type = type; must be the 2nd statement.

So, /*INSERT-1*/ must be replaced with:

super(capacity); this.type = type;

Correction Working with Inheritance - 2

A - Comment the statement at Line 9

class A is declared public and defined in com.training.oca package, there is no problem in accessing class A outside com.training.oca package.

class B is defined in com.training.oca.test package, to extend from class A either use import statement “import com.training.oca.A;” or fully qualified name of the class com.training.oca.A. No issues with this class definition as well.

Variable i1 is declared public in class A, so Line 8 doesn’t cause any compilation error. Variable i2 is declared protected so it can only be accessed in subclass using inheritance but not using object reference variable. obj.i2 causes compilation failure.

class B inherits variable i2 from class A, so inside class B it can be accessed by using either this or super. Line 10 and Line 11 don’t cause any compilation error.

Correction Working with Inheritance - 3

A - ClassCastException is thrown at runtime

Class A and B are declared public and inside same package com.training.oca. Method print() of class A has correctly been overridden by B.

print() method is public so no issues in accessing it anywhere.

Let’s check the code inside main method.

A obj1 = new A(); => obj1 refers to an instance of class A.

B obj2 = (B)obj1; => obj1 is of type A and it is assigned to obj2 (B type), hence explicit casting is necessary. obj1 refers to an instance of class A, so at runtime obj2 will also refer to an instance of class A. sub type can’t refer to an object of super type so at runtime B obj2 = (B)obj1; will throw ClassCastException.

Correction Working with Inheritance - 4

C - Replace the code at Line 12 with: abstract class NewsPaper extends Paper {

D - Insert at Line 14: public void setOrientation() {}

First you should find out the reason for compilation error. Methods declared in Printable interface are implicitly abstract, no issues with Printable interface.

class Paper is declared abstract and it implements Printable interface, it overrides setMargin() method but setOrientation() method is still abstract. No issues with class Paper as it is an abstract class and can have 0 or more abstract methods.

class NewsPaper is concrete class and it extends Paper class (which is abstract). So class NewsPaper must override setOrientation() method OR it must be declared abstract.

Replacing Line 9 with ‘public abstract void setOrientation();’ is not necessary and it will not resolve the compilation error in NewsPaper class.

Replacing Line 7 with ‘class Paper implements Printable {’ will cause compilation failure of Paper class as it inherits abstract method ‘setOrientation’.

Correction Working with Inheritance - 5

C - public

A top level interface can be declared with either public or default modifiers.

public interface is accessible across all packages but interface declared with default modifier and be accessed in the defining package only.

Correction Working with Inheritance - 6

D - Compilation error for Child(int, int) Constructor

super(); inside Parent(int) constructor invokes the no-arg constructor of Object class and hence no compilation error for Parent(int) constructor.

super(0); inside Child(int) constructor invokes Parent(int) constructor, which is available and hence no issues.

Child(int, int) constructor has both super(i) and this(j) statements. A constructor should have super(…) or this(…) but not both. Hence Child(int, int) causes compilation failure.

As all the classes are defined in Test.java file under com.training.oca.test package, hence child.i and child.j don't give compilation error. i and j are declared with package scope.

Correction Working with Inheritance - 7

C - final

Class declared as final can’t be inherited. Examples are: String, Integer, System etc.

Correction Working with Inheritance - 8

C - Inner Peace!

System.out.println(new Child()); invokes the toString() method on Child’s instance.

Parent class’s method can be invoked by super keyword. super.toString() method returns “Inner” and “Inner”.concat(“Peace!”) returns “Inner Peace!”.

Correction Working with Inheritance - 9

B - Add the public modifier to the help() method of LogHelper class

C - Remove the protected modifier from the help() method of Helper class

E - Add the protected modifier to the help() method of LogHelper class

Let us first find out the issue:

As instance variables are hidden by subclasses and not overridden, hence overriding rules are not for the instance variables. There are no issues with variables ‘num’ and ‘operation’.

log() method is declared with default modifier in both the classes, hence no issue with log() method as well.

abstract method help() is declared with protected modifier in Helper class and in LogHelper class, it is overridden with default modifier and this causes compilation error. So below solutions to resolved this issue:

1. Remove the protected modifier from the help() method of Helper class: Both the overridden and overriding methods will have same default modifier, which is allowed

OR

2. Add the protected modifier to the help() method of LogHelper class: Both the overridden and overriding methods will have same protected modifier, which is allowed

OR

3. Add the public modifier to the help() method of LogHelper class: Overridden method will have protected modifier and overriding method will have public modifier, which is allowed

Correction Working with Inheritance - 10

A - Compilation error

class Car doesn’t extend from Vehicle class, this means Vehicle is not super type of Car.

Hence, Vehicle obj = new Car(); causes compilation error.

Correction Working with Inheritance - 11

D - B b = new A();

B b = new A(); -> child class reference cannot refer to parent class object. This will give compilation error.

A a = new B(); -> parent class reference can refer to child class object. This is Polymorphism.

B a = new B(); -> No issues at all.

A a = new A(); -> No issues at all.

Correction Working with Inheritance - 12

B - MNO

M ^ N ^ O [obj refers to instance of O class] ^ P

obj instanceof M -> true

obj instanceof N -> true

obj instanceof O -> true

but

obj instanceof P -> false

Correction Working with Inheritance - 13

D - Method m1() in Implementer class is not implemented correctly.

void m1(); in interface I01 is equivalent to public abstract void m1(); So method m1() is implicitly public and abstract.

In java, a class can extend from only one class but can implement multiple interfaces. Correct keywords are: extends and implements.

so class declaration is correct.

As method m1() is implicitly public in I01, hence overriding method in Implementer class should also be public. But it is protected and hence compiler complains.

Correction Working with Inheritance - 14

B - Compilation Error

super(); is added by the compiler as the first statement in both the constructors.

Class Super extends from Object class and Object class has no-argument constructor, hence no issues with the constructor of Super class.

But no-arg constructor is not available in Super class, hence calling super(); from Sub class constructor gives compilation error.

Correction Working with Inheritance - 15

A - final class Electronics {}

class Dog {}: can be sub-classed within the same package.

abstract class Cat {}: can be sub-classed within the same package.

final class Electronics {}: a class with final modifier cannot be sub-classed.

private class Car {}: a top level class cannot be declared with private modifier.

Correction Working with Inheritance - 16

A - refer to parent class object.

super refers to parent class object and this refers to currently executing object.

Correction Working with Inheritance - 17

D - None of the other options

Derived class overrides method m1() of Base class. Access modifier of method m1() in Base class is protected, so overriding method can use protected or public.

But overriding method in this case used default modifier and hence there is compilation error.

Correction Working with Inheritance - 18

C - ClassCastException is thrown at runtime

Class A, B and C are declared public and inside same package com.training.oca. Method print() of class A has correctly been overridden by B and C.

print() method is public so no issues in accessing it anywhere.

Let’s check the code inside main method.

A obj1 = new C(); => obj1 refers to an instance of C class, it is polymorphism.

A obj2 = new B(); => obj2 refers to an instance of B class, it is polymorphism.

C obj3 = (C)obj1; => obj1 actually refers to an instance of C class, so at runtime obj3 (C type) will refer to an instance of C class. As obj1 is of A type so explicit typecasting is necessary.

C obj4 = (C)obj2; => obj2 actually refers to an instance of B class, so at runtime obj4 (C type) will refer to an instance of B class. B and C are siblings and can’t refer to each other, so this statement will throw ClassCastException at runtime.

Correction Working with Inheritance - 19

D -

```java
class Mosquito extends Insect implements Flyable{}
```

A class in Java extends class and implements interface.

Correction Working with Inheritance - 20

B -

```
Replace /*INSERT-1*/ with:
this.type = type;
Replace /*INSERT-2*/ with:
this(type);
```

Java compiler adds super(); as the first statement inside constructor, if call to another constructor using this(…) or super(…) is not available.

Compiler adds super(); as the first line in Word's constructor: Word(int pages, String type) { super(); } but Document class doesn't have a no-argument constructor and that is why Word's constructor `Word(int pages, String type)` causes compilation error.

Word(String) constructor is actually not setting the passed type argument. Replace /*INSERT-1*/ with: `this.type = type;` will set the value to type variable.

As the first statement inside Word(int pages, String type){} constructor, you can either have `super(pages);` or `this(type);` but not both.

Replacing /*INSERT-2*/ with `super(pages);` will be redundant as in the next statement `super.pages = pages;`, pages variable of Document class is set. Hence, replacing /*INSERT-2*/ with `this(type);` is needed to set the type variable.

Correction Working with Inheritance - 21

A - Animal Deer

Cat class doesn’t override the jump() method of Animal class, in fact jump(int) method is overloaded in Cat class.

Deer class overrides jump() method of Animal class.

Reference variable cat is of Animal type, cat.jump() syntax is fine and as Cat doesn’t override jump() method hence Animal version is invoked, which prints Animal to the console.

Even though reference variable deer is of Animal type but at runtime deer.jump(); invokes overriding method of Deer class, this prints Deer to the console.

Correction Working with Inheritance - 22

A - N M

Default constructor added by Java compiler in B class is:

B() { super(); }

On executing new B(); statement, class B’s default constructor is invoked, which invokes no-arg constructor of class A [super();].

no-arg constructor of class A invokes parameterized constructor of class A [this(1);].

N is printed first and after that M is printed.

Correction Working with Inheritance - 23

E - Code compiles successfully and on execution prints DERIVED on to the console

print() method at Line n2 hides the method at Line n1. So, no compilation error at Line n2.

Reference variable ‘b’ is of type Base, so `(Derived) b` does not cause any compilation error. Moreover, at runtime it will not throw any ClassCastException as well because b is null. Had ‘b’ been referring to an instance of Base class [Base b = new Base();], `(Derived) b` would have thrown ClassCastException.

d.print(); doesn’t cause any compilation error but as this syntax creates confusion, so it is not a good practice to access the static variables or static methods using reference variable, instead class name should be used. Derived.print(); is the preferred syntax.

d.print(); invokes the static print() method of Derived class and prints DERIVED on to the console.

Correction Working with Inheritance - 24

A - 0:4

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n3 hides the variable created at Line n1 and Line n4 overrides the getSide() method of Line n2. There is no compilation error for Square class as it correctly overrides getSide() method. You can use any access modifier at Line n3 as well, there are no rules for variable hiding.

‘s’ is of Shape type, hence s.side equals to 0 and s.getSide() invokes overriding method of Square class and it returns 4. Hence output is: 0:4.

Correction Working with Inheritance - 25

F - Compilation error in Dog.java file

super(); is added by the compiler as the first statement in both the constructors:

```
Animal() {
    super();
    System.out.print("ANIMAL-");
}
```

and

```
public Dog() {
    super();
    System.out.print("DOG");
}
```

Class Animal extends from Object class and Object class has no-argument constructor, hence no issues with the constructor of Animal class.

Animal class's constructor has package scope, which means it is accessible to all the classes declared in package 'a'. But Dog class is declared in package 'b' and hence `super();` statement inside Dog class's constructor causes compilation error as no-argument constructor of Animal class is not visible.

There is no compilation error in Test.java file as Dog class's constructor is public and therefore `new Dog();` compiles successfully.

Correction Working with Inheritance - 26

G - It prints KEEP_GOING_ on to the console

Super class defines a method with name Super() but not any constructor. Hence compiler adds below default constructor in Super class:

```
Super() {
    super();
}
```

Class Super extends from Object class and Object class has no-argument constructor, which is called by the super(); statement in above default constructor.

Java compiler also adds `super();` as the first statement inside the no-argument constructor of Base class:

```
Base() {
    super();
    Super();
    System.out.print("GOING_");
}
```

As Base extends Super and both the classes are in the same package, hence `super();` invokes the no-argument constructor of Super class and `Super();` invokes the Super() method of Super class. Base class inherits the Super() method of Super class.

No compilation error in any of the classes.

On executing Test class, main(String[]) is invoked, which executes `new Base();` statement.

No-argument constructor of Base class is invoked, which executes `super();`, hence no-argument constructor of Super class is invoked.

Next, `Super();` is executed and this invokes the Super() method of Super class and hence KEEP_ is printed on to the console.

After that, `System.out.print("GOING_");` is executed and GOING_ is printed on to the console.

main(String []) method finishes its execution and program terminates successfully after printing KEEP_GOING_ on to the console.

Correction Working with Inheritance - 27

E - It compiles successfully and on execution prints below:

```
Good Night!
Good Night!
Good Night!
```

Variable x is of X type (superclass) and refers to an instance of Z type (subclass).

At Line n1, compiler checks whether greet() method is available in class X or not. As greet() method is available in class X, hence no compilation error for Line n1.

At Line n2, x is casted to Y and compiler checks whether greet() method is available in class Y or not. As greet() method is available in class Y, hence no compilation error for Line n2.

At Line n3, x is casted to Z and compiler checks whether greet() method is available in class Z or not. As greet() method is available in class Z, hence no compilation error for Line n3.

There is no compilation error in the given code it compiles successfully.

Variable x refers to an instance of Z class and at Line n1, n2 and n3 same instance is being used. Which overriding method to invoke, is decided at runtime based on the instance.

At runtime, all three statements, at Line n1, Line n2 and Line n3 would invoke the greet() method of Z class, which would print Good Night! three times on to the console.

Correction Working with Inheritance - 28

A - An exception is thrown at runtime

Class M and M are declared public and inside same package com.udayankhattry.oca. Method printName() of class M has correctly been overridden by N.

printName() method is public so no issues in accessing it anywhere.

Let’s check the code inside main method.

M obj1 = new M(); => obj1 refers to an instance of class M.

N obj2 = (N)obj1; => obj1 is of type M and it is assigned to obj2 (N type), hence explicit casting is necessary. obj1 refers to an instance of class M, so at runtime obj2 will also refer to an instance of class M. sub type can’t refer to an instance of super type so at runtime `N obj2 = (N)obj1;` will throw ClassCastException.

Correction Working with Inheritance - 29

G - gogogo

Class Q correctly overrides the compute(String) method of P class and class R correctly overrides the compute(String) method of Q class. Keyword super is used to invoke the method of parent class.

At Line n1, reference variable ‘p’ refers to an instance of class R, hence p.compute(“Go”) invokes the compute(String) method of R class.

return super.compute(str.replace(‘o’, ‘O’)); => return super.compute(“Go”.replace(‘o’, ‘O’)); => return super.compute(“GO”);

It invokes the compute(String) method of Parent class, which is Q.

=> return super.compute(str.toLowerCase()); => return super.compute(“GO”.toLowerCase()); => return super.compute(“go”);

It invokes the compute(String) method of Parent class, which is P.

=> return str + str + str; => return “gogogo”;

Control goes back to compute(String) method of Q and to the compute(String) method of R, which returns “gogogo”.

Line n2 prints gogogo on to the console.

Correction Working with Inheritance - 30

A - Line n2 causes compilation error

The static method of subclass cannot hide the instance method of superclass. static main(String []) method at Line n2 tries to hide the instance main(String []) method at Line n1 and hence Line n2 causes compilation error.

There is no issue with Line n3 as it is a valid syntax to invoke the instance main(String []) method of M class.

No issue with Line n4 as well as it correctly invokes static main(String []) method of N class.

Correction Working with Inheritance - 31

B - Delete the Line n2

C - Change Line n4 to Parent obj = new Child();

F - Change Line n5 to System.out.printin(obj.getVar());

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n2 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

Line n5 causes compilation error as obj is of Child type and ‘var’ is declared private in Child class. Variable ‘var’ of Child class cannot be accessed outside the Child class.

Let’s check all the options one by one:

‘Change Line n1 to private int var = 1000;’ => It will not rectify the existing error of Line n5, in fact after this change, Line n3 will also cause compilation error.

‘Delete the Line n2’ => After deleting this line, obj.var at Line n5 will refer to variable ‘var’ of Parent class. Hence, output will be 1000 in this case.

‘Change Line n3 to return var;’ => This will have no effect to the output of the code, as getVar() method has not been invoked.

‘Change Line n4 to Parent obj = new Child();’ => After this modification, obj becomes Parent type, hence obj.var will refer to variable ‘var’ of Parent class. Hence, output will be 1000 in this case.

‘Delete the method getVar() from the Child class’ => This will have no effect to the output of the code, as getVar() method has not been invoked.

‘Change Line n5 to System.out.println(obj.getVar());’ => obj.getVar() will invoke the getVar() method of Child class and this method returns the variable value from Parent class (super.var). Hence, output will be 1000 in this case.

Correction Working with Inheritance - 32

A - Remove the protected access modifier from the getCalories() method of Food class

E - Make the getCalories() method of JunkFood class protected

F - Make the getCalories() method of JunkFood class public

abstract methods cannot be declared with private modifier as abstract methods need to be overridden in child classes.

abstract methods can be declared with either public, protected and package (no access modifier) modifier and hence overriding method cannot be declared which private modifier in the child class. That is why getCalories() method in Food and JunkFood classes cannot be declared private.

Access modifier of overriding method should either be same as the access modifier of overridden method or it should be less restrictive than the access modifier of overridden method. Hence below solutions will work:

1. Remove the protected access modifier from the getCalories() method of Food class: By doing this, both the overridden and overriding methods will have same access modifier (no access modifier)

or

2. Make the getCalories() method of JunkFood class protected: By doing this, both the overridden and overriding methods will have same access modifier (protected)

or

3. Make the getCalories() method of JunkFood class public: By doing this, access modifier of overriding method (which is public) is less restrictive than the access modifier of overridden method (which is protected)

Correction Working with Inheritance - 33

D - class Test compiles successfully and on execution prints X3 on to the console

As per Java 8, default methods were added in the interface. Interface X1 defines default method print(), there is no compilation error in interface X1. Method print() is implicitly public in X1.

interface X2 extends X1 and it overrides the default method print() of X1, overriding method in X2 is implicitly abstract and public. An interface in java can override the default method of super type with abstract modifier. interface X2 compiles successfully.

interface X3 extends X2 and it implements the abstract method print() of X2, overriding method in X3 is default and implicitly public. An interface in java can implement the abstract method of super type with default modifier. interface X3 compiles successfully.

class X implements X3 and therefore it inherits the default method print() defined in interface X3.

`X1 obj = new X();` compiles successfully as X1 is of super type (X implements X3, X3 extends X2 and X2 extends X1).

`obj.print();` invokes the default method print() defined in interface X3 and hence X3 is printed on to the console.

Correction Working with Inheritance - 34

A - There is a compilation error in Dog.java file

Method move() declared in Moveable interface is implicitly public and abstract.

Abstract class Animal has non-abstract method move() and it is declared with no modifier (package scope). Abstract class in java can have 0 or more abstract methods. Hence Animal class compiles successfully.

class Dog extends Animal and as both the classes Animal and Dog are within the same package ‘com.training.oca’, Dog inherits the move() method defined in Animal class.

Dog class implements Moveable interface as well, therefore it must implement public move() method as well. But as inherited move() method from Animal class is not public, therefore Dog class fails to compile.

Correction Working with Inheritance - 35

G - It compiles successfully and on execution prints £35.0 on to the console

default methods were added in Java 8. Class Chair correctly implements Sellable interface and it also overrides the default symbol() method of Sellable interface.

At Line n1, ‘obj’ refers to an instance of Chair class, so obj.symbol() and obj.getPrice() invoke the overriding methods of Chair class only.

obj.symbol() returns “£” and obj.getPrice() returns 35.0

At Line n2, ‘+’ operator behaves as concatenation operator and Line n2 prints £35.0 on to the console.

Correction Working with Inheritance - 36

A - Test class compiles successfully and on execution prints 1010 on to the console

Variable ‘count’ declared inside interface Counter is implicitly public, static and final. Line n1 compiles successfully.

Line n2 creates one dimensional array of 2 elements of Counter type and both the elements are initialized to null. Line n2 compiles successfully.

Though correct way to refer static variable is by using the type name, such as Counter.count but it can also be invoked by using Counter reference variable. Hence ctr.count at Line n3 correctly points to the count variable at Line n1.

For invoking static fields, object is not needed, therefore even if ‘ctr’ refers to null, ctr.count doesn’t throw NullPoionterException. Given loop executes twice and therefore output is: 1010

Correction Working with Inheritance - 37

D - Replace /*INSERT*/ with below code:

```
public double profit() {
  return Profitable2.super.profit();
}
```

Profit class causes compilation error as it complains about duplicate default methods: Profitable1.profit() and Profitable2.profit(). To rectify this error abstract class Profit must override the profit() method.

default keyword for method is allowed only inside the interface and default methods are implicitly public. So overriding method should use public modifier and shouldn't use default keyword.

If you want to invoke the default method implementation from the overriding method, then the correct syntax is: [Interface_name].super.[default_method_name].

Hence, `Profitable1.super.profit();` will invoke the default method of Profitable1 interface and `Profitable2.super.profit();` will invoke the default method of Profitable2 interface.

Based on above points, let's check all the options one by one:

No need for any modifications, code compiles as is: ✗

Replace /*INSERT*/ with below code:

```
double profit() {
    return 50.0;
}: ✗
```

profit() method must be declared with public access modifier.

Replace /*INSERT*/ with below code:

```
public default double profit() {
    return 50.0;
}: ✗
```

default keyword for method is allowed only inside the interface.

Replace /*INSERT*/ with below code:

```
protected double profit() {
    return 50.0;
}: ✗
```

profit() method must be declared with public access modifier.

Replace /*INSERT*/ with below code:

```
public double profit() {
    return Profitable1.profit();
}: ✗
```

Profitable1.profit(); causes compilation error as correct syntax is: Profitable1.super.profit();

Replace /*INSERT*/ with below code:

```
public double profit() {
    return Profitable2.super.profit();
}: ✓
```

It compiles successfully.

Correction Working with Inheritance - 38

D - There is a compilation error in MyLogger class

As per Java 8, default and static methods were added in the interface. Interface ILog defines static method log(), there is no compilation error in interface ILog.

Abstract class Log defines the static log() method. Abstract class can have 0 or more abstract methods. Hence, no compilation error in class Log as well.

Default methods of an interface are implicitly public and are inherited by the implementer class. Class MyLogger implements ILog interface and therefore it inherits the default log() method of ILog interface.

Also, the scope of static log() method of abstract class Log is not limited to class Log only but MyLogger also gets Log.log() method in its scope.

So, MyLogger class has instance method log() [inherited from ILog interface] and static method log() [from Log class] and this causes conflict. Static and non-static methods with same signature are not allowed in one scope, therefore class Log fails to compile.

Correction Working with Inheritance - 39

A - Test class compiles successfully and on execution prints SUPER on to the console

Variable ‘name’ declared inside interface Super is implicitly public, static and final. Line n1 compiles successfully.

In Java a class can extend from only one class but an interface can extend from multiple interfaces. Line n2 compiles successfully.

Variable ‘name’ can be accessed in 2 ways: Super.name and Sub.name.

Though correct way to refer static variable is by using the type name, such as Sub.name but it can also be invoked by using Sub reference variable. Hence, sub.name at Line n3 correctly points to the name variable at Line n1.

For invoking static fields, object is not needed, therefore even if sub refers to null, sub.name doesn’t throw NullPoionterException.

Test class compiles successfully and on execution prints SUPER on to the console.

Correction Working with Inheritance - 40

A - None of the other options

super.open(); => Using super keyword, you can access methods and variables of immediate parent class, hence if you replace /*INSERT*/ with `super.open();`, then open() method of Padlock class will be invoked.

super.super.open(); => super.super is not allowed in java, it causes compilation error.

((Lock)super).open(); => Not possible to cast super keyword in java, it causes compilation error.

(Lock)super.open(); => super.open(); will be evaluated first as dot (.) operator has higher precedence than cast. super.open(); returns void and hence it cannot be casted to Lock. It also causes compilation error.

In fact, it is not possible to directly reach to 2 levels, super keyword allows to access methods and variables of immediate parent class only (just 1 level up). Hence, correct answer is: ‘None of the other options’

Correction Working with Inheritance - 41

C - Test class compiles successfully and on execution prints 42.0 on to the console

‘profitPercentage’ variable of Profitable interface is implicitly public, static and final.

Line n1 defines the instance variable ‘profitPercentage’ of Business class. There is no error at Line n1.

Super type reference variable can refer to an instance of Sub type, therefore no issues at Line n2 as well.

Even though correct syntax for accessing interface variable is by using Interface name, such as Profitable.profitPercentage but reference variable also works. obj.profitPercentage doesn’t cause any compilation error.

As, obj is of Profitable type, hence obj.profitPercentage points to the ‘profitPercentage’ variable of Profitable type. Given code compiles successfully and on execution prints 42.0 on to the console.

Correction Working with Inheritance - 42

B - Three statements

/*INSERT*/ cannot be replaced with interface as work() method at Line n1 is neither abstract nor default. Hence, statements 3 and 4 will not work.

Let’s check other statements:

1. abstract class Work implements Workable: abstract class in java can have 0 or more abstract methods. It compiles successfully.

2. class Work implements Workable: It correctly implements the work() method of Workable interface, hence it compiles successfully.

3. abstract class Work: abstract class in java can have 0 or more abstract methods. It compiles successfully.

Hence, out of 5 statements, 3 will compile successfully.

Correction Working with Inheritance - 43

D - There is a compilation error in Test.java file

As per Java 8, default and static methods were added in the interface. There is no issue in Shrinkable.java file.

class AntMan implements Shrinkable interface but as there is no abstract method in Shrinkable interface, hence AntMan class is not needed to implement any method. AntMan.java file compiles successfully.

static method of Shrinkable interface can only be accessed by using Shrinkable.shrinkPercentage(). `AntMan.shrinkPercentage();` causes compilation error.

Correction Working with Inheritance - 44

B - There is a compilation error at Line n2

F - There is a compilation error at Line n4

Variable 'salePercentage' declared inside interface Buyable is implicitly public, static and final. As per Java 8, default and static methods were added in the interface. There is no compilation error in Buyable.java file.

class Book implements Buyable interface but as there is no abstract method in Buyable interface, hence Book class is not needed to implement any method. Book.java file compiles successfully.

`Buyable [] arr = new Buyable[2];` creates one dimensional array of 2 elements of Buyable type and both the elements are initialized to null.

There are some difference in which static variables and static methods of the interface are accessed.

Correct and only way to access static method of an Interface is by using the name of the interface, such as Buyable.salePercentage(). Line n2 and Line n4 cause compilation error.

As far as public static final variable of interface is concerned, even through the correct way to access static variable is by using the name of the interface, such as Buyable.salePercentage but it can also be accessed by using following:

Reference variable of the interface: Buyable obj1 = null; System.out.println(obj1.salePercentage);

Name of the implementer class: System.out.println(Book.salePercentage);

Reference variable of the implementer class: Book obj2 = null; System.out.println(obj2.salePercentage);

Hence, Line n1 and Line n3 compile successfully.

Correction Working with Inheritance - 45

A - Compilation error in class Test

Class Y correctly extends class X and it overrides method A() and provides two new methods B() and C().

At Line n1, obj is of X type and therefore obj.B(); and obj.C(); cause compilation error as these methods are not defined in class X.

Correction Working with Inheritance - 46

D - Line n1 causes compilation error

As per Java 8, default and static methods were added in the interface. Interface M defines static method log(), there is no compilation error in interface M.

Also the scope of static log() method of M is limited to interface M and it can be invoked by using Interface name only, M.log().

Abstract class A also defines the static log() method. Abstract class can have 0 or more abstract methods. Hence, no compilation error in class A as well.

Super type reference variable can refer to an instance of Sub type, therefore the statement `M obj1 = new MyClass();` compiles successfully.

obj1 is of M type, hence `obj1.log();` tries to tag the static method of M but static log() method of M can only be invoked by using M.log();.

Therefore, Line n1 causes compilation error.

Scope of static log() method of A is not limited to class A only but MyClass also gets A.log() method in its scope.

There are different ways in which static method of an abstract class can be accessed:

1. By using the name of the abstract class: A.log(); //Preferred way
2. By using the reference variable of abstract class: A o1 = null; o1.log();
3. By using the name of the subclass: MyClass.log();
4. By using the reference variable of the subclass: MyClass o2 = null; o2.log();

Hence, Line n2 and Line n3 compile successfully.

Correction Working with Inheritance - 47

D - Compilation Error in Sub class

Parent (Super) class constructor is invoked by `super();` (all letters in lowercase) from within the constructor of subclass.

First statement inside no-argument constructor of Sub class is: `Super();` (Letter ‘S’ is in uppercase) and hence it causes compilation error.

Correction Working with Inheritance - 48

D - Class Word causes compilation error (Correct)

As per Java 8, default methods were added in the interface. Interface Document defines default method getType(), there is no compilation error in interface Document. Method getType() is implicitly public in Document.

interface WordDocument extends Document and it overrides the default method getType() of Document, overriding method in WordDocument is implicitly abstract and public. An interface in java can override the default method of super type with abstract modifier. interface WordDocument compiles successfully.

class Word implements WordDocument and as WordDocument interface has abstract method getType(), and as class Word doesn’t implement the getType() method hence it causes compilation failure.

Correction Working with Inheritance - 49

C -

```
-:$
-:€
```

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n3 hides the variable created at Line n1 and Line n4 overrides the getNotation() method of Line n2. There is no compilation error for USDollar class as it correctly overrides getNotation() method.

Similarly, Line n5 hides the variable created at Line n1 and Line n6 overrides the getNotation() method of Line n2. There is no compilation error for Euro class as it correctly overrides getNotation() method as well.

‘c1’ is of Currency type, hence c1.notation refers to “-” and c1.getNotation() invokes overriding method of USDollar class and it returns “$”.

Similarly, c2.notation refers to “-” and c2.getNotation() invokes overriding method of Euro class and it returns “€”.

Correction Working with Inheritance - 50

B - It executes successfully and prints INHALE-INHALE-EXHALE

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n2 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

At Line n3, obj1 is of Base type and refers to an instance of Base class.

At Line n4, obj2 is of Base type and refers to an instance of Derived class.

At Line n5, as obj2 refers to an instance of Derived class, hence typecasting it to Derived type doesn't cause any Exception. obj3 is of Derived type and refers to an instance of Derived class.

Let's check the expression of Line n6:

obj1.msg + "-" + obj2.msg + "-" + obj3.msg;

=> (obj1.msg + "-") + obj2.msg + "-" + obj3.msg; //+ operator is left to right associative and behaves as concatenation operator as one of the operand is of String type.

=> ((obj1.msg + "-") + obj2.msg) + "-" + obj3.msg;

=> (((obj1.msg + "-") + obj2.msg) + "-") + obj3.msg;

Let's solve the expression now:

=> ((("INHALE" + "-") + obj2.msg) + "-") + obj3.msg; //obj1 is of Base type, hence obj1.msg refers to "INHALE"

=> (("INHALE-" + obj2.msg) + "-") + obj3.msg;

=> (("INHALE-" + "INHALE") + "-") + obj3.msg; //obj2 is of Base type, hence obj2.msg refers to "INHALE"

=> ("INHALE-INHALE" + "-") + obj3.msg;

=> "INHALE-INHALE-" + obj3.msg;

In above expression, left operand is of String type and right operand is of Object type, so toString() method is invoked. So, given expression is evaluated as:

=> "INHALE-INHALE-" + obj3.msg.toString();

=> “INHALE-INHALE-” + “EXHALE”; //As obj3.msg is of Object type and refers to an instance of String type, hence toString() method on “EXHALE” instance is invoked and this returns “EXHALE”.

=> “INHALE-INHALE-EXHALE”;

Line n7 prints INHALE-INHALE-EXHALE on to the console.

Correction Working with Inheritance - 51

B - ((Rideable)horse).ride("“emma');

D - ((Horse)(Rideable)horse).ride(“emma');

F - ((Rideable)(Horse)horse).ride(“EMMA”);

G - ((Horse)horse).ride(“Emma'”);

Let's check all the options one by one:

horse.ride(“EMMA”); ✗ Variable ‘horse' is of Animal type and ride(String) method is not defined in Animal class, therefore it causes compilation error.

(Horse)horse.ride(“EMMA”); ✗ horse.ride(“EMMA”) will be evaluated first as dot (.) operator has higher precedence than cast. horse.ride(“EMMA”) returns void, hence it cannot be casted to Horse type. This would cause compilation error.

((Horse)horse).ride(“Emma”); ✓ Variable ‘horse' refers to an instance of Horse type and variable ‘horse' is casted to Horse type. Horse class has ride(String) method, hence no compilation error. ride(String) method of Horse class will get invoked at runtime and will print the expected output. As, name.toUpperCase() method is invoked, hence it doesn't matter in what case you pass the name, in the output name will always be displayed in the upper case.

((Rideable)horse).ride(“emma”); ✓ Variable ‘horse' refers to an instance of Horse type and variable ‘horse' is casted to Rideable type (super type of Horse). Rideable interface has ride(String) method, hence no compilation error. ride(String) method of Horse class will get invoked at runtime and will print the expected output.

(Rideable)(Horse)horse.ride(“EMMA”); ✗ horse.ride(“EMMA”) will be evaluated first as dot (.) operator has higher precedence than cast. horse.ride(“EMMA”) returns void, hence it cannot be casted to Horse type. This would cause compilation error.

(Horse)(Rideable)horse.ride(“EMMA”); ✗ horse.ride(“EMMA”) will be evaluated first as dot (.) operator has higher precedence than cast. horse.ride(“EMMA”) returns void, hence it cannot be casted to Rideable type. This would cause compilation error.

((Rideable)(Horse)horse).ride(“EMMA”); ✓ Variable ‘horse' refers to an instance of Horse type, it is first casted to Horse type and then casted to Rideable type. Rideable interface has ride(String) method, hence no compilation error. ride(String) method of Horse class will get invoked at runtime and will print the expected output.

((Horse)(Rideable)horse).ride(“emma”); ✓ Variable ‘horse’ refers to an instance of Horse type, it is first casted to Rideable type and then casted to Horse type. Horse class has ride(String) method, hence no compilation error. ride(String) method of Horse class will get invoked at runtime and will print the expected output.

Correction Working with Inheritance - 52

D - ((Parent)(Child)gc).quote

As instance variables are hidden by subclasses and not overridden, therefore instance variable can be accessed by using explicit casting.

Let’s check all the options one by one:

gc.quote => It refers to “PLAY PLAY PLAY” as gc is of GrandChild class.

(Parent)gc.quote => gc.quote will be evaluated first as dot (.) operator has higher precedence than cast. gc.quote refers to String, hence it cannot be casted to Parent type. This would cause compilation error.

((Parent)gc).quote => Variable ‘gc’ is casted to Parent type, so this expression refers to “MONEY DOESN’T GROW ON TREES”. It is one of the correct options.

((Parent)(Child)gc).quote => ‘gc’ is of GrandChild type, it is first casted to Child and then to Parent type and finally quote variable is accessed, so this expression refers to “MONEY DOESN’T GROW ON TREES”. It is also one of the correct options.

(Parent)(Child)gc.quote => gc.quote will be evaluated first as dot (.) operator has higher precedence than cast. gc.quote refers to String, hence it cannot be casted to Child type. This would cause compilation error.

Correction Working with Inheritance - 53

C - Given code compiles successfully and on execution prints below in the output:

```
Flying at 20 degrees.
Landing at -20 degrees.
```

As per Java 8, default and static methods were added in the interface and default methods can invoke static method as well. Hence, there is no issue with the Flyable interface.

class Aeroplane implements Flyable interface, hence it inherits the default method fly() and static method horizontalDegree() can be accessed using Flyable.horizontalDegree(). It also provides the implementation of land() method. There is no issue with Aeroplane class as well.

On execution below text is printed on to the console:

```
Flying at 20 degrees.
Landing at -20 degrees.
```

Correction Working with Inheritance - 54

E - Line n3 causes compilation error

Variable ‘count’ declared inside interface GetSetGo is implicitly public, static and final. Line n1 compiles successfully.

Line n2 creates one dimensional array of 5 elements of GetSetGo type and all 5 elements are initialized to null. Line n2 compiles successfully.

Though correct way to refer static variable is by using the type name, such as GetSetGo.count but it can also be invoked by using GetSetGo reference variable. Hence, obj.count at Line n3 correctly points to the count variable at Line n1. But as variable ‘count’ is implicitly final, therefore obj.count++ causes compilation error. Line n3 fails to compile.

Line n4 compiles successfully as variable ‘count’ is implicitly static and GetSetGo.count is the correct syntax to point to ‘count’ variable of interface GetSetGo.

Correction Working with Inheritance - 55

E - It executes successfully and prints 102 on to the console

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n2 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

‘obj’ is of Super type, hence obj.num refers to num variable at Line n1, which is of String type.

Expression at Line n3:

obj.num += 2

=> obj.num = obj.num + 2

=> obj.num = “10” + 2

=> obj.num = “102”

obj.num refers to “102” and same is printed on to the console.

Correction Working with Inheritance - 56

A - Compilation error

Constructors cannot use final, abstract or static modifiers. As no-argument constructor of MySubClass uses final modifier, therefore it causes compilation error.

Correction Working with Inheritance - 57

A - Given code compiles successfully and on execution Test class prints 2 on to the console

As per Java 8, default methods were added in the interface. Interface Perishable1 defines default method maxDays(), there is no compilation error in interface Perishable1. Method maxDays() is implicitly public in Perishable1.

interface Perishable2 extends Perishable1 and it overrides the default method maxDays() of Document, overriding method in Perishable2 is implicitly public. Interface Perishable2 compiles successfully.

Class Milk implements Perishable2 and Perishable1. Although it is redundant for Milk class to implement Preishable1 as Perishable2 already extends Perishable1.

There is no conflict in Milk class as it inherits the default method maxDays() of Perishable2 interface. Milk class compiles successfully.

`Perishable1 obj = new Milk();` It compiles fine as Perishable1 is supertype and Milk is subtype.

`obj.maxDays()` executes the default maxDays() method of Perishable2 interface and it returns 2. `System.out.println(obj.maxDays());` prints 2 on to the console.

Correction Working with Inheritance - 58

D - Line n4 causes compilation error

Variable ‘i’ declared inside interface I1 is implicitly public, static and final and similarly variable i declared inside interface I2 is implicitly public, static and final as well.

In Java a class can extend from only one class but an interface can extend from multiple interfaces. static variables are not inherited and hence there is no issue with Line n1.

I1.i points to variable ‘i’ of interface I1.

I2.i points to variable ‘i’ of interface I2.

I3.i is an ambiguous call as compiler is not sure whether to point to I1.i or I2.i and therefore, Line n4 causes compilation error.

Correction Working with Inheritance - 59

C - Compilation error in RuledPaper class

Instance method of subclass cannot override the static method of superclass.

Instance method at Line n2 tries to override the static method at Line n1 and hence Line n2 causes compilation error.

There is no issue with Line n3 as reference variable of superclass can refer to an instance of subclass.

At Line n4, paper.getType() doesn’t cause compilation error but as this syntax creates confusion, so it is not a good practice to access the static variables or static methods using reference variable, instead class name should be used. Paper.getType() is the preferred syntax.

Correction Working with Inheritance - 60

H - -1

Method can have same name as that of the Class. Hence, void Base() is a valid method declaration in Base class.

Line n2 invokes the Base() method and not the constructor.

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n5 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

Line n7 correctly overrides the Base() method of class Base.

Compiler adds super(); as the 1st statement inside the no-argument constructor of Base class and Derived class.

There is no compilation error, so let’s check the execution.

new Derived() at Line n9 invokes the constructor of Base class, at this point instance variable id is declared and 0 is assigned to it. In fact, instance variable id of Base class is also declared and 0 is assigned to it. Compiler added super(); as the first statement inside this constructor, hence control goes to the no-argument constructor of Base class.

Compiler added super(); as the first statement inside this constructor as well, hence it invokes the no-argument constructor of the Object class. No-argument constructor of Object class finishes its execution and control goes back to the constructor of Base class. Before it starts executing remaining statements inside the constructor, instance variable assignment statement (if available) are executed. This means 1000 is assigned to variable id of Base class.

Line n2 is executed next, Base() method defined in Derived class is executed. Which overriding method to invoke, is decided at runtime based on the instance. Instance is of Derived class (because of Line n9), hence control starts executing Base() method of Derived class.

Line n8 is executed next, Derived class hides the id variable of Base class and that is why at Line n8, id points to variable created at Line n5. This id variable still stores the value 0 as Base class’s constructor has not finishes its execution.

value of id is decremented by 1, so id becomes -1 and -1 is printed on to the console. Base() method finishes its execution and control goes back to Line n2. No-argument constructor of Base class finishes its execution and control goes back to the constructor of Derived class. Before it starts executing remaining statements inside the constructor, instance variable assignment statement (if available) are executed. This means 2000 is assigned to variable id of Base class.

No-argument constructor of Derived class finishes its execution and control goes back to Line n9. main(String []) method finishes its execution and program terminates successfully.

Hence, output is -1.

Correction Working with Inheritance - 61

C - Class Test causes compilation error

As per Java 8, default and static methods were added in the interface. Interface I1 defines static method print(String), there is no compilation error in interface I1.

Also the scope of print(String) method of I1 is limited to interface I1 and it can be invoked by using Interface name only, I1.print(““).

class C1 implements I1 and it also defines print(String) instance method. Even though class C1 implements I1, it doesn’t have static print(String) method in its scope, therefore class C1 compiles successfully.

Super type reference variable can refer to an instance of Sub type, therefore the statement `I1 obj = new C1();` compiles successfully.

obj is of I1 type, hence `obj.print("Java");` tries to tag the static method of I1 but static print(String) method of I1 can only be invoked by using I1.print(“Java”);.

Therefore, `obj.print("Java");` causes compilation error.

Correction Working with Inheritance - 62

C - Replace /*INSERT*/ with Object NUM;

D - Replace /*INSERT*/ with short NUM;

G - Remove final modifier from Line n1

H - Replace /*INSERT*/ with int NUM;

Variable NUM is declared in Super class and class Sub extends Super, hence NUM can be accessed by using obj.NUM.

But as NUM Is final, hence it cannot be reassigned, therefore Line n2 causes compilation error. Let's check all the options one by one:

Remove final modifier from Line n1 => ✓ Valid option and in this case output is 200.

Replace /*INSERT*/ with byte NUM; => ✗ In this case, class Sub hides the variable NUM of Super class but Line n2 will still not compile as byte range is from -128 to 127 and 200 is out of range value.

Replace /*INSERT*/ with short NUM; => ✓ In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to short type. In this case output is 200.

Replace /*INSERT*/ with int NUM; => In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to int type. In this case output is 200.

Replace /*INSERT*/ with float NUM; => ✗ In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to float type. But output in this case will be 200.0 and not 200.

Replace /*INSERT*/ with double NUM; => ✗ In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to double type. But output in this case will be 200.0 and not 200.

Replace /*INSERT*/ with boolean NUM; => ✗ In this case, class Sub hides the variable NUM of Super class but Line n2 will still not compile as boolean type in java allows 2 values true and false. 200 is not compatible with boolean type.

Replace /*INSERT*/ with Object NUM; => ✓ In this case, class Sub hides the variable NUM of Super class and at Line n2, value 200 is boxed to Integer, which is then assigned to obj.NUM. So, obj.NUM refers to an instance of

Integer class. Line n3 invokes toString() method of Integer class and hence 200 is printed on to the console.

Correction Working with Inheritance - 63

B - Line n3 causes compilation error

CommunicationLog class overrides count() and get() methods of Log class.

There are 2 rules related to return types:

1. If return type of overridden method is of primitive type, then overriding method should use same primitive type.

2. If return type of overridden method is of reference type, then overriding method can use same reference type or its sub-type (also known as covariant return type).

count() method at Line n1 returns long but overriding method at Line n3 returns int and that is why Line n3 causes compilation error.

get() method at Line n2 returns Object but overriding method at Line n4 returns String. String is a subclass of Object, so it is a case of covariant return type and hence allowed. Line n4 compiles successfully.